jg-you / noisy-networks-measurements

Code · Issues · Pull requests · Actions · Projects · Security · Insights

master

noisy-networks-measurements / examples / **readme.md**

jg-you Restructured dir

History

1 contributor

Raw · Blame

21 lines (11 sloc) 2.12 KB

# Template list

## Models obtained by modifying the `undir.stan` template

- `poisson_data_ER_prior.stan` : Takes a symmetric matrix of number of observations `X` as input. Models `X[i,j]` as drawn from a Poisson variable, with success rate determined by presence or absence of the edge `(i,j)`. Uses an Erdös-Rényi prior.
- `bernoulli_data_ER_prior.stan` : Takes as input a symmetric matrix of number of observations `X`, and a total number of observations `M`. Models `X[i,j]` as the outcome of `M` draws, with success rate determined by presence or absence of the edge `(i,j)`. Uses an Erdös-Rényi prior.
- `poisson_data_SCM_prior.stan` : Same as Poisson data model above, but uses a soft configuration model as the network prior.
- `bernoulli_data_SCM_prior.stan` : Same as Bernoulli data model above, but uses a soft configuration model as the network prior.

## Models obtained by modifying the `undir_multitype.stan` template

- `multitype_poisson_data_ER_prior.stan` : Takes a symmetric matrix of number of observations `X` as input. Models `X[i,j]` as drawn from a Poisson variable, with success rate determined by presence or absence of the edge `(i,j)`. **Uses multiple edge types**. Uses a generalized Erdös-Rényi prior.

## Accelerated versions (no templates)

- `fast_poisson.stan` : Accelerated version of `poisson.stan`. Takes two vectors `X` and `Y` as input. The entry `X[i]` of `X` gives the number of pairs of nodes that are measured as interacting `Y[i]` times. Return the matrix `Q` in vector format matching `X` and `Y`.
- `fast_bernoulli.stan` : Accelerated version of `bernoulli.stan`. Takes as input two vectors `X` and `Y`, and a total number of observations `M`. The entry `X[i]` of `X` gives the number of pairs of nodes that are measured as interacting `Y[i]` times, out of `M` measurements. Return the matrix `Q` in vector format matching `X` and `Y`.

jg-you / noisy-networks-measurements

<> Code | Issues | Pull requests | Actions | Projects | Security | Insights

master

noisy-networks-measurements / examples / bernoulli_data_ER_prior.stan

jg-you Restructured dir

History

1 contributor

Raw | Blame

48 lines (43 sloc) 1.29 KB

```
data {
  int<lower=1> n;
  int<lower=0> X[n, n];
  int M;
  real<lower=0> rates_prior[2];
  real<lower=0> rho_prior[2];
}
parameters {
  positive_ordered[2] rates;
  real<lower=0, upper=1> rho;
}
model {
  rates ~ beta(rates_prior[1], rates_prior[2]);
  rho ~ beta(rho_prior[1], rho_prior[2]);

  for (i in 1:n) {
    for (j in i + 1:n) {
      real log_mu_ij_0 = binomial_lpmf(X[i, j] | M, rates[1]);
      real log_mu_ij_1 = binomial_lpmf(X[i, j] | M, rates[2]);

      real log_nu_ij_0 = bernoulli_lpmf(0 | rho);
      real log_nu_ij_1 = bernoulli_lpmf(1 | rho);

      real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
      real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
      if (z_ij_0 > z_ij_1) {target += z_ij_0 + log1p_exp(z_ij_1 - z_ij_0);}
      else {target += z_ij_1 + log1p_exp(z_ij_0 - z_ij_1);}
    }
  }
}
generated quantities {
  real Q[n ,n];
  for (i in 1:n) {
    Q[i, i] = 0;
    for (j in i+1:n) {
      real log_mu_ij_0 = binomial_lpmf(X[i, j] | M, rates[1]);
      real log_mu_ij_1 = binomial_lpmf(X[i, j] | M, rates[2]);

      real log_nu_ij_0 = bernoulli_lpmf(0 | rho);
      real log_nu_ij_1 = bernoulli_lpmf(1 | rho);

      real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
      real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
      Q[i, j] = 1 / (1  + exp(z_ij_0 - z_ij_1));
      Q[j, i] = Q[i, j];
    }
  }
}
```

jg-you / noisy-networks-measurements

<> Code   Issues   Pull requests   Actions   Projects   Security   Insights

master

noisy-networks-measurements / examples / bernoulli_data_SCM_prior.stan

jg-you Remove unused parameter from models   History

1 contributor

Raw   Blame

50 lines (45 sloc)   1.36 KB

```
data {
  int<lower=1> n;
  int<lower=0> X[n, n];
  int M;
  real<lower=0> rates_prior[2];
}
parameters {
  positive_ordered[2] rates;
  simplex[n] lambda;
  real<lower = 0> scale;
}
model {
  rates ~ beta(rates_prior[1], rates_prior[2]);
  scale ~ exponential(100);

  for (i in 1:n) {
    for (j in i + 1:n) {
      real log_mu_ij_0 = binomial_lpmf(X[i, j] | M, rates[1]);
      real log_mu_ij_1 = binomial_lpmf(X[i, j] | M, rates[2]);

      real r = inv_logit(scale * lambda[i] * lambda[j]);
      real log_nu_ij_0 = bernoulli_lpmf(0 | r);
      real log_nu_ij_1 = bernoulli_lpmf(1 | r);

      real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
      real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
      if (z_ij_0 > z_ij_1) {target += z_ij_0 + log1p_exp(z_ij_1 - z_ij_0);}
      else {target += z_ij_1 + log1p_exp(z_ij_0 - z_ij_1);}
    }
  }
}
generated quantities {
  real Q[n ,n];
  for (i in 1:n) {
    Q[i, i] = 0;
    for (j in i+1:n) {
      real log_mu_ij_0 = binomial_lpmf(X[i, j] | M, rates[1]);
      real log_mu_ij_1 = binomial_lpmf(X[i, j] | M, rates[2]);

      real r = inv_logit(scale * lambda[i] * lambda[j]);
      real log_nu_ij_0 = bernoulli_lpmf(0 | r);
      real log_nu_ij_1 = bernoulli_lpmf(1 | r);

      real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
      real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
      Q[i, j] = 1 / (1  + exp(z_ij_0 - z_ij_1));
      Q[j, i] = Q[i, j];
    }
  }
}
```

jg-you / noisy-networks-measurements

<> Code  Issues  Pull requests  Actions  Projects  Security  Insights

master

noisy-networks-measurements / examples / poisson_data_ER_prior.stan

jg-you Restructured dir  History

1 contributor

Raw  Blame

48 lines (43 sloc)  1.31 KB

```stan
data {
  int<lower=1> n;
  int<lower=0> X[n, n];
  real<lower=0> rates_std_prior[2];
  real<lower=0> rho_prior[2];
}
parameters {
  positive_ordered[2] rates;
  real<lower=0, upper=1> rho;
}
model {
  rates[1] ~ normal(1, rates_std_prior[1]);
  rates[2] ~ normal(1, rates_std_prior[2]);
  rho ~ beta(rho_prior[1], rho_prior[2]);

  for (i in 1:n) {
    for (j in i + 1:n) {
      real log_mu_ij_0 = poisson_lpmf(X[i, j] | rates[1]);
      real log_mu_ij_1 = poisson_lpmf(X[i, j] | rates[2]);

      real log_nu_ij_0 = bernoulli_lpmf(0 | rho);
      real log_nu_ij_1 = bernoulli_lpmf(1 | rho);

      real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
      real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
      if (z_ij_0 > z_ij_1) {target += z_ij_0 + log1p_exp(z_ij_1 - z_ij_0);}
      else {target += z_ij_1 + log1p_exp(z_ij_0 - z_ij_1);}
    }
  }
}
generated quantities {
  real Q[n ,n];
  for (i in 1:n) {
    Q[i, i] = 0;
    for (j in i+1:n) {
      real log_mu_ij_0 = poisson_lpmf(X[i, j] | rates[1]);
      real log_mu_ij_1 = poisson_lpmf(X[i, j] | rates[2]);

      real log_nu_ij_0 = bernoulli_lpmf(0 | rho);
      real log_nu_ij_1 = bernoulli_lpmf(1 | rho);

      real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
      real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
      Q[i, j] = 1 / (1  + exp(z_ij_0 - z_ij_1));
      Q[j, i] = Q[i, j];
    }
  }
}
```

jg-you / noisy-networks-measurements

<> Code | Issues | Pull requests | Actions | Projects | Security | Insights

master

noisy-networks-measurements / examples / poisson_data_SCM_prior.stan

jg-you Remove unused parameter from models

History

1 contributor

Raw | Blame

50 lines (45 sloc) 1.38 KB

```
data {
  int<lower=1> n;
  int<lower=0> X[n, n];
  real<lower=0> rates_std_prior[2];
}
parameters {
  positive_ordered[2] rates;
  simplex[n] lambda;
  real<lower = 0> scale;
}
model {
  rates[1] ~ normal(1, rates_std_prior[1]);
  rates[2] ~ normal(1, rates_std_prior[2]);
  scale ~ exponential(100);

  for (i in 1:n) {
    for (j in i + 1:n) {
      real log_mu_ij_0 = poisson_lpmf(X[i, j] | rates[1]);
      real log_mu_ij_1 = poisson_lpmf(X[i, j] | rates[2]);

      real r = inv_logit(scale * lambda[i] * lambda[j]);
      real log_nu_ij_0 = bernoulli_lpmf(0 | r);
      real log_nu_ij_1 = bernoulli_lpmf(1 | r);

      real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
      real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
      if (z_ij_0 > z_ij_1) {target += z_ij_0 + log1p_exp(z_ij_1 - z_ij_0);}
      else {target += z_ij_1 + log1p_exp(z_ij_0 - z_ij_1);}
    }
  }
}
generated quantities {
  real Q[n ,n];
  for (i in 1:n) {
    Q[i, i] = 0;
    for (j in i+1:n) {
      real log_mu_ij_0 = poisson_lpmf(X[i, j] | rates[1]);
      real log_mu_ij_1 = poisson_lpmf(X[i, j] | rates[2]);

      real r = inv_logit(scale * lambda[i] * lambda[j]);
      real log_nu_ij_0 = bernoulli_lpmf(0 | r);
      real log_nu_ij_1 = bernoulli_lpmf(1 | r);

      real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
      real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
      Q[i, j] = 1 / (1  + exp(z_ij_0 - z_ij_1));
      Q[j, i] = Q[i, j];
    }
  }
}
```

jg-you / noisy-networks-measurements

<> Code | Issues | Pull requests | Actions | Projects | Security | Insights

master

noisy-networks-measurements / examples / multitype_poisson_data_ER_prior.stan

jg-you Restructured dir | History

1 contributor

Raw | Blame

62 lines (59 sloc) 1.25 KB

```
data {
  int<lower=1> n;
  int<lower=0> X[n, n];
  int T; // number of edge types
  real<lower=0> rates_std_prior[T];
}
parameters {
  positive_ordered[T] rates;
  simplex[T] rho;
}
model {
  for (k in 1:T)
  {
    rates[k] ~ normal(1, rates_std_prior[k]);
  }

  for (i in 1:n) {
    for (j in i + 1:n) {
      vector[T] z_ij;
      vector[T] z_max_vector;
      real z_max;
      for (k in 1:T) {
        real log_mu_ij_k = poisson_lpmf(X[i, j] | rates[k]);
        real log_nu_ij_k = log(rho[k]);

        z_ij[k] = log_mu_ij_k + log_nu_ij_k;
      }
      z_max = max(z_ij);
      z_max_vector = rep_vector(z_max, T);
      target += z_max + log_sum_exp(z_ij - z_max_vector);
    }
  }
}
generated quantities {
  real Q[n ,n, T];
  for (i in 1:n) {
    for (k in 1:T) {
      Q[i, i, k] = 0;
    }
    for (j in i+1:n) {
      vector[T] z_ij;
      real accu;
      for (k in 1:T)
      {
        real log_mu_ij_k = poisson_lpmf(X[i, j] | rates[k]);
        real log_nu_ij_k = log(rho[k]);

        z_ij[k] =  log_mu_ij_k + log_nu_ij_k;
      }
      for (k in 1:T)
      {
        accu = 0;
        for (k_prime in 1:T)
        {
          accu += exp(z_ij[k_prime] - z_ij[k]);
        }
        Q[i, j, k] = 1 / accu;
        Q[j, i, k] = Q[i, j, k];
      }
    }
  }
}
```

jg-you / noisy-networks-measurements

<> Code | Issues | Pull requests | Actions | Projects | Security | Insights

master

noisy-networks-measurements / examples / compiler.py

jg-you Update email

History

1 contributor

Raw | Blame

48 lines (41 sloc) 1.64 KB

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Python functions to interact with the stan models.

Author:
Jean-Gabriel Young <jean.gabriel.young@gmail.com>
"""
import numpy as np
import pickle
import pystan
import os

abs_path = os.path.dirname(os.path.abspath(__file__))


def compile_stan_model(model_file, force=False):
    """Autocompile Stan model."""
    model_name = os.path.splitext(model_file)[0]
    source_path = os.path.join(abs_path, model_name + '.stan')
    target_path = os.path.join(abs_path, model_name + '.bin')

    if os.path.exists(target_path):
        # Test whether the model has changed and only compile if it did
        with open(target_path, 'rb') as f:
            current_model = pickle.load(f)
        with open(source_path, 'r') as f:
            file_content = "".join([line for line in f])
        if file_content != current_model.model_code or force:
            print(target_path, "[Compiling]", ["", "[Forced]"][force])
            model = pystan.StanModel(source_path, model_name=model_name)
            with open(target_path, 'wb') as f:
                pickle.dump(model, f)
        else:
            print(target_path, "[Skipping --- already compiled]")
    else:
        # If model binary does not exist, compile it
        print(target_path, "[Compiling]")
        model = pystan.StanModel(source_path, model_name=model_name)
        with open(target_path, 'wb') as f:
            pickle.dump(model, f)


if __name__ == '__main__':
    print("Compiling models with pystan version", pystan.__version__)
    for m in os.listdir("examples"):
        if os.path.splitext(m)[1] == ".stan":
            compile_stan_model(m, force=False)
```

jg-you / noisy-networks-measurements

<> Code | Issues | Pull requests | Actions | Projects | Security | Insights

master

noisy-networks-measurements / examples / fast_bernoulli.stan

jg-you Typos

History

1 contributor

Raw | Blame

47 lines (42 sloc) 1.3 KB

```
data {
  int n;    // number of non-empty pairs class
  int X[n]; // number of pairs of nodes in each classes
  int Y[n]; // number of observations for each class
  int M;    // Total number of interactions
  // Priors
  real<lower=0> rates_prior[2];
  real<lower=0> rho_prior[2];
}
parameters {
  positive_ordered[2] rates;
  real<lower=0, upper=1> rho;
}
model {
  rates ~ beta(rates_prior[1], rates_prior[2]);
  rho ~ beta(rho_prior[1], rho_prior[2]);

  for (i in 1:n)
  {

    real log_mu_i_0 = binomial_lpmf(Y[i] | M, rates[1]);
    real log_mu_i_1 = binomial_lpmf(Y[i] | M, rates[2]);
    real log_nu_i_0 = bernoulli_lpmf(0 | rho);
    real log_nu_i_1 = bernoulli_lpmf(1 | rho);

    real z = 0;
    real z_i_0 = log_mu_i_0 + log_nu_i_0;
    real z_i_1 = log_mu_i_1 + log_nu_i_1;
    if (z_i_0 > z_i_1) {z += z_i_0 + log1p_exp(z_i_1 - z_i_0);}
    else {z += z_i_1 + log1p_exp(z_i_0 - z_i_1);}
    target += X[i] * z;
  }
}
generated quantities {
  real Q[n];
  for (i in 1:n) {
    real log_mu_i_0 = binomial_lpmf(Y[i] | M, rates[1]);
    real log_mu_i_1 = binomial_lpmf(Y[i] | M, rates[2]);
    real log_nu_i_0 = bernoulli_lpmf(0 | rho);
    real log_nu_i_1 = bernoulli_lpmf(1 | rho);
    real z_i_0 = log_mu_i_0 + log_nu_i_0;
    real z_i_1 = log_mu_i_1 + log_nu_i_1;
    Q[i] = 1 / (1 + exp(z_i_1 - z_i_0));
  }
}

```

jg-you Typos

History

1 contributor

Raw | Blame

47 lines (42 sloc) 1.28 KB

```
data {
  int n;    // number of non-empty pairs class
  int X[n]; // number of pairs of nodes in each classes
  int Y[n]; // number of observations for each class
  // Priors
  real<lower=0> rates_std_prior[2];
  real<lower=0> rho_prior[2];
}
parameters {
  positive_ordered[2] rates;
  real<lower=0, upper=1> rho;
}
model {
  rates[1] ~ normal(1, rates_std_prior[1]);
  rates[2] ~ normal(1, rates_std_prior[2]);
  rho ~ beta(rho_prior[1], rho_prior[2]);

  for (i in 1:n)
  {

    real log_mu_i_0 = poisson_lpmf(Y[i] | rates[1]);
    real log_mu_i_1 = poisson_lpmf(Y[i] | rates[2]);
    real log_nu_i_0 = bernoulli_lpmf(0 | rho);
    real log_nu_i_1 = bernoulli_lpmf(1 | rho);

    real z = 0;
    real z_i_0 = log_mu_i_0 + log_nu_i_0;
    real z_i_1 = log_mu_i_1 + log_nu_i_1;
    if (z_i_0 > z_i_1) {z += z_i_0 + log1p_exp(z_i_1 - z_i_0);}
    else {z += z_i_1 + log1p_exp(z_i_0 - z_i_1);}
    target += X[i] * z;
  }
}
generated quantities {
  real Q[n];
  for (i in 1:n) {
    real log_mu_i_0 = poisson_lpmf(Y[i] | rates[1]);
    real log_mu_i_1 = poisson_lpmf(Y[i] | rates[2]);
    real log_nu_i_0 = bernoulli_lpmf(0 | rho);
    real log_nu_i_1 = bernoulli_lpmf(1 | rho);
    real z_i_0 = log_mu_i_0 + log_nu_i_0;
    real z_i_1 = log_mu_i_1 + log_nu_i_1;
    Q[i] = 1 / (1 + exp(z_i_1 - z_i_0));
  }
}
```

Model templates, that can be used to implement custom models without writing boilerplate code. Portions of the code that must be modified are marked with double square brackets, like so: `[[ ]]`. The available templates are:

- `undir.stan` : Generic template for measurement of undirected graph.
- `dir.stan` : Generic template for measurement of directed graph.
- `undir_multitype.stan` : Generic template for measurement of undirected graph with many edge types.

See `stan` reference manual for details on how to use various probability mass functions and probability density functions.

jg-you / noisy-networks-measurements

<> Code   Issues   Pull requests   Actions   Projects   Security   Insights

master

noisy-networks-measurements / templates / dir.stan

jg-you Restructured dir   History

1 contributor

Raw   Blame

60 lines (55 sloc)   1.4 KB

```
data {
  int<lower=1> n;
  int<lower=0> X[n, n];
  // [[Additional data go here]]
}
parameters {
  // [[Parameters go here]]
  //
  // For example:
  // real<lower=0,upper=1> theta;
}
model {
  // [[Priors go here]]
  //
  // For example:
  // theta ~ beta(1, 1/2);

  for (i in 1:n) {
    for (j in 1:n) {
      if (i != j) {
        // [[Data model goes here]]
        real log_mu_ij_0 = ;
        real log_mu_ij_1 = ;

        // [[Network model goes here]]
        real log_nu_ij_0 = ;
        real log_nu_ij_1 = ;

        // Boilerplate code below, do not modify
        real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
        real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
        if (z_ij_0 > z_ij_1) {target += z_ij_0 + log1p_exp(z_ij_1 - z_ij_0);}
        else {target += z_ij_1 + log1p_exp(z_ij_0 - z_ij_1);}
      }
    }
  }
}
generated quantities {
  // Generate edge probability matrix
  real Q[n ,n];
  for (i in 1:n) {
    Q[i, i] = 0;
    for (j in 1:n) {
      if (i != j) {
        // [[Data model goes here, as in model block]]
        real log_mu_ij_0 = ;
        real log_mu_ij_1 = ;

        // [[Network model goes here, as in model block]]
        real log_nu_ij_0 = ;
        real log_nu_ij_1 = ;

        // Boilerplate code below, do not modify
        real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
        real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
        Q[i, j] = 1 / (1  + exp(z_ij_0 - z_ij_1));
      }
    }
  }
}
```

jg-you / noisy-networks-measurements

Code Issues Pull requests Actions Projects Security Insights

master

noisy-networks-measurements / templates / undir.stan

jg-you Restructured dir

History

1 contributor

Raw Blame

57 lines (52 sloc) 1.34 KB

```
data {
  int<lower=1> n;
  int<lower=0> X[n, n];
  // [[Additional data go here]]
}
parameters {
  // [[Parameters go here]]
  //
  // For example:
  // real<lower=0,upper=1> theta;
}
model {
  // [[Priors go here]]
  //
  // For example:
  // theta ~ beta(1, 1/2);

  for (i in 1:n) {
    for (j in i + 1:n) {
      // [[Data model goes here]]
      real log_mu_ij_0 = ;
      real log_mu_ij_1 = ;

      // [[Network model goes here]]
      real log_nu_ij_0 = ;
      real log_nu_ij_1 = ;

      // Boilerplate code below, do not modify
      real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
      real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
      if (z_ij_0 > z_ij_1) {target += z_ij_0 + log1p_exp(z_ij_1 - z_ij_0);}
      else {target += z_ij_1 + log1p_exp(z_ij_0 - z_ij_1);}
    }
  }
}
generated quantities {
  // Generate edge probability matrix
  real Q[n ,n];
  for (i in 1:n) {
    Q[i, i] = 0;
    for (j in i+1:n) {
      // [[Data model goes here, as in model block]]
      real log_mu_ij_0 = ;
      real log_mu_ij_1 = ;

      // [[Network model goes here, as in model block]]
      real log_nu_ij_0 = ;
      real log_nu_ij_1 = ;

      // Boilerplate code below, do not modify
      real z_ij_0 = log_mu_ij_0 + log_nu_ij_0;
      real z_ij_1 = log_mu_ij_1 + log_nu_ij_1;
      Q[i, j] = 1 / (1  + exp(z_ij_0 - z_ij_1));
      Q[j, i] = Q[i, j];
    }
  }
}
```

jg-you / **noisy-networks-measurements**

noisy-networks-measurements / templates / **undir_multitype.stan**

Raw Blame

70 lines (67 sloc) 1.46 KB

```
data {
  int<lower=1> n;
  int<lower=0> X[n, n];
  int T; // number of edge types
  // [[Additional data go here]]
}
parameters {
  // [[Parameters go here]]
  //
  // For example:
  // real<lower=0,upper=1> theta;
}
model {
  // [[Priors go here]]
  //
  // For example:
  // theta ~ beta(1, 1/2);

  for (i in 1:n) {
    for (j in i + 1:n) {
      vector[T] z_ij;
      vector[T] z_max_vector;
      real z_max;
      for (k in 1:T) {
        // [[Data model goes here]]
        real log_mu_ij_k = ;
        // [[Network model goes here]]
        real log_nu_ij_k = ;

        // Boilerplate code below, do not modify
        z_ij[k] = log_mu_ij_k + log_nu_ij_k;
      }
      z_max = max(z_ij);
      z_max_vector = rep_vector(z_max, T);
      target += z_max + log_sum_exp(z_ij - z_max_vector);
    }
  }
}
generated quantities {
  // Generate edge probability matrix
  real Q[n ,n, T];
  for (i in 1:n) {
    for (k in 1:T) {
      Q[i, i, k] = 0;
    }
    for (j in i+1:n) {
      vector[T] z_ij;
      real accu;
      for (k in 1:T)
      {
        // [[Data model goes here]]
        real log_mu_ij_k = ;
        // [[Network model goes here]]
        real log_nu_ij_k = ;

        // Boilerplate code below, do not modify
        z_ij[k] =  log_mu_ij_k + log_nu_ij_k;
      }
      for (k in 1:T)
      {
        accu = 0;
        for (k_prime in 1:T)
        {
          accu += exp(z_ij[k_prime] - z_ij[k]);
        }
        Q[i, j, k] = 1 / accu;
      }
    }
  }
}
```